IT a anatomie firmy

(Mikroslužby)

(pracovní dokument)

MBI tým, Lumír Srch (ITS)

VŠE Praha, 2024

Obsah

1. Úvod

Dokument představuje pouze ***stručné shrnutí principů, výhod a nevýhod*** jednoho z perspektivních konceptů vývoje aplikací, konceptu založeného na mikroslužbách. Text dokumentu vychází z části práce (Aschmann, 2020) a verifikovaný v praxi partnerských firem. Navazuje na širší kontext v dokumentu [Řízení IT].

Mikroslužby se chápou jako aktuální vývojová ***etapa ve vytváření aplikací***. Umožňují efektivně, ***agilně vyvíjet aplikace***, operativně v nich ***reagovat na změny*** uživatelských požadavků, a to při ***snižování nákladů*** na vývoj. Tendence směřují k distribuovaným systémům se specifickou funkcionalitou založených právě na využití mikroslužeb a orchestračních platforem jako např. Kubernetes.

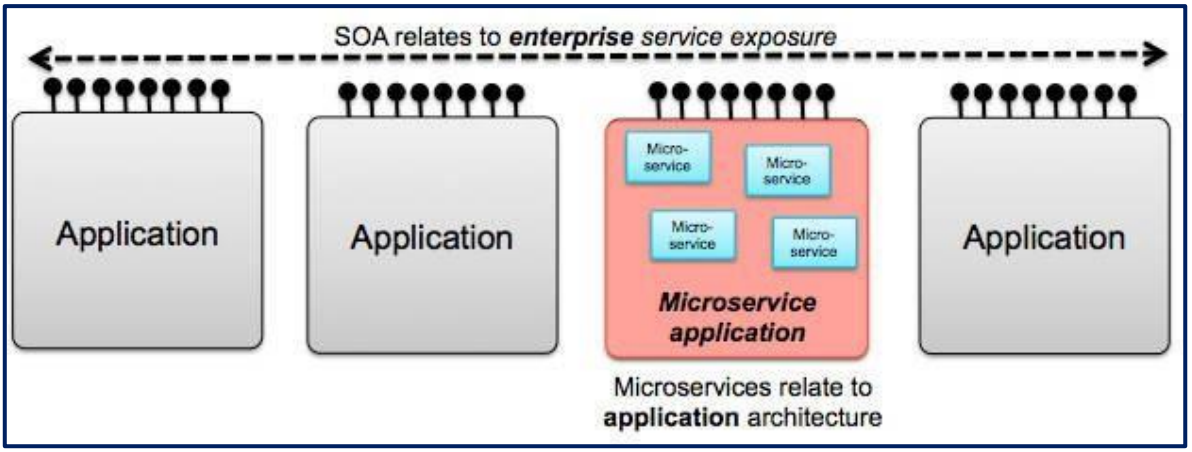
Architektura mikroslužeb umožňuje rozdělení aplikací na ***malé izolované nezávisle fungující části***, což umožňuje operativně vyměňovat jednotlivé celky bez vlivu na celou aplikaci. S rostoucím počtem mikroslužeb je stále náročnější je komplexně spravovat a udržovat. Pro tyto účely vznikla technologie ***service mesh,*** která sleduje komunikaci mezi službami, šifruje a dešifruje zprávy mezi službami, zajišťuje autorizaci a autentizaci zpráv a poskytuje load balancing.

2. Mikroslužby, základní principy

IBM definuje mikroslužby jako přístup, kde je **aplikace složená z volně spojených a nezávislých malých služeb.** Tyto služby mají svůj vlastní zdroj dát, komunikují spolu pomocí ***REST*** rozhraní a jsou uspořádané podle jejich business významu. Výhodami jsou zejména jednoduché upravování kódu a funkcionality, využití libovolného programovacího jazyka a škálovatelnost komponent nezávisle na ostatních. (IBM Cloud Education, 2019).

Je třeba vymezit ***rozdíly mezi SOA a mikroslužbami***. Podstatné jak pro SOA, tak pro mikroslužby je to, že obojí směřují ke stejnému účelu, tj. rozpad částí aplikace na komponenty, nezávislost mezi službami a standardizované komunikační protokoly. ***SOA*** se ale především zaměřuje na vystavování rozhraní aplikací, aby data i aplikace byly efektivně přístupné. ***Architektura mikroslužeb*** se orientuje na aplikaci. Umožňuje ji členit na malé nezávise vyměnitelné časti. Nedefinuje, jak mají spolu aplikace komunikovat. Jde o princip struktury aplikací pro dosažení jejich potřebné agility a dostupnosti. (Clark, 2016)

SOA je koncept a způsob vytváření softwaru, které ***využívají dostupné služby na síti***. Každá služba představuje ***ucelenou byznys funkcionalitu***, např. jedna obchodní transakce, která je oddělená od okolí a poskytuje rozhraní pro komunikaci mezi službami. To mohou být jednoduché zprávy mezi dvěma službami nebo komplexnější orchestrace více služeb na bázi protokolů Simple Object Access Protocol (SOAP) anebo Representational State Transfer (REST). Služby se mohou ***vytvářet znovu nebo využitím původních funkcionalit služeb*** a jejich standardních rozhraní. Tím je také možné ***přesněji definovat požadavky*** byznysu v podobě jednotlivých funkcí služeb. (IBM Cloud Education, 2019)



Obrázek 2-1: Rozdíly mezi mikroslužbami a SOA (Clark, 2016)

Obrázek 2-1 představuje rozdíly mezi mikroslužbami a SOA. SOA znamená řešení architektury na úrovni celé firmy, mikroslužby jsou zaměřené na jednu aplikaci a její strukturu.

3. Výhody mikroslužeb

Mikroslužby jsou rovněž založené na využití virtualizace, cloud computingu, agilních metodik a dalších. Mikroslužby dosahují svých výhod zejména díky využívání konceptů distribuovaných systémů a SOA. (Newman, 2015). Mezi hlavní výhody mikroslužeb patří:

- ***Agilnost a produktivita*** – umožňují využívat nové technologie v některých komponentách a neovlivňovat ostatní. Mohou být upravované, automatizovaně testované a nasazované na sobě nezávisle, to znamená, že se mění pouze konkrétní komponenta, aniž se zasahuje do ostatních. Takto nezávisle lze testovat i nové technologie.
- ***Odolnost*** – díky nezávislosti, chyba v komponentě neznamená obvykle výpadek celé aplikace. Je možné chybnou mikroslužbu izolovat, vrátit k původní funkční verzi a chod celé aplikace zachovat.
- ***Nezávislé nasazování služeb*** – malé úpravy kódu už nemusí představovat dlouhou dobu příprav a testování. Nová funkcionalita může být nasazená rychle bez velkých zásahů do celé aplikace a bez značných rizik. V případě problémů je možné se operativně vrátit k původní verzi.
- ***Nahraditelnost*** – rozsah kódu individuální mikroslužby je v rozmezí několika desítek nebo stovek řádek kódu. Díky tomu je časově nenáročné ji nahradit novou nebo ji zrušit.
- ***Samostatnost*** – mikroslužba je nezávislá entita, která může být nasazená jako izolovaná služba na platformě nebo jako samotný proces. Komunikace probíhá výhradně přes API, event. pomocí streamingu anebo messagingu s ohledem na jasnou separaci služeb. *(Newman, 2015).*
- ***Správný nástroj*** – s mikroslužbami mají všechny komponenty vlastní zdroje dat a tím je možné zajistit jejich jednodušší vývoj s nejnovějšími technologiemi.
- ***Škálovatelnost*** – schopnost škálování se s mikroslužbami maximalizuje, kterákoliv komponenta může být nezávisle multiplikovaná nebo zredukovaná v počtu instancí podle v daném okamžiku požadované výpočetní síly.

4. Problémy mikroslužeb

Mikroslužby s sebou nesou i jistá omezení, na které třeba rovněž upozornit. K těm hlavním patří (Richardson, 2019):

- ***Dekompozice aplikace*** – neexistuje přesný algoritmus, resp. návod na rozdělení systému na jednotlivé služby. Pokud je systém rozdělený špatně, může to znamenat vytvoření distribuovaného monolitu složeného z provázaných služeb, které ale musí být vždy nasazované spolu.
- ***Volba správného času*** – čas přechodu nastane až tehdy, kdy se začne řešit problematika komplikovanosti aplikace. Až tehdy je třeba ji funkčně rozdělit na mikroslužby.
- ***Komplexnost mikroslužeb*** – implementace, které jdou napříč větším počtem služeb vyžadují nové přístupy. Vlastní zdroje dat pro každou mikroslužbu mohou znamenat jisté komplikace při vytváření nového transakčního systému. Každá transakce aktualizuje záznam v databázi a následně pošle správu nebo aktivuje spínač pro následující transakci.
- K ***operačním složitostem*** patří mnohé verze a typy služby, které musí být provozované současně.
- **Koordinace nasazování** – je nutné mít sestavený plán nasazování mikroslužeb, kde je specifikováno jejich pořadí, v kterém mají být nasazovány. Toto pořadí je založené na základě závislostí mezi danými službami. S tím souvisí verzování API, které tento problém řeší.

Mikroslužby se postupně stávají ***všeobecně přijímaným přístupem v řešení IT aplikací***. Je to komplikovaný technologický přístup. Analytické zhodnocení jejich přístupů pomůže v jejich rozšíření a efektivním uplatnění v praxi.

5. Docker

Podle (Aschmann, 2020, upraveno)

Docker je ***open-source platforma***, která umožňuje izolovat ***aplikace do kontejnerů***, tedy spustitelných komponent, které ***kombinují operační systém, knihovny, samotnou aplikaci a další součásti*** potřebné pro funkce aplikace. Jsou ovládané přes příkazovou řádku a ***ulehčují instalaci provoz i odstraňování softwaru***. Výhodou Dockeru je jeho ***jednoduchost*** vytváření, nasazení a správy kontejnerů.

Jedná se o omezenou verzi operačního systému, která obsahuje pouze nutné komponenty pro provoz dané aplikace a s tím souvisejí výhody jako ***izolace aplikací, škálovatelnost a dále zejména***: (IBM Cloud Education, 2020)

- ***Nenáročnost*** – kontejnery obsahují pouze nutné části operačního systému a jsou tak nenáročné na výpočetní kapacity. Znamená to i rychlou inicializaci po nasazení do provozu.
- ***Efektivita*** – díky izolaci procesů operačního systému a virtualizace se realizuje sdílení zdrojů mezi aplikacemi a s tím úspory na infrastruktuře.
- ***Produktivita*** – vývoj aplikací je efektivnější a produktivnější.

Docker pomáhá sestavovat kontejnery díky ***konzistentnímu postupu jejich tvorby*** a využitím existujících kontejnerových nástrojů. (Nickoloff, 2019). Docker vyvinul vlastní technologii, s těmito možnostmi (IBM Cloud Education, 2020):

- ***Přenositelnost*** – kontejnery Docker je možné spustit na různých počítačích v datových centrech nebo v cloudu.
- ***Jednoduchost*** – Docker povoluje jeden proces na kontejner.
- ***Automatizace*** – Docker podporuje automatické vytváření kontejnerů na základě zdrojového kódu aplikace.
- ***Verze*** – Docker sleduje historii verzí kontejnerů a umožňuje návrat ke starším verzím a zobrazuje změny mezi jednotlivými verzemi.
- ***Opětovné využití kontejnerů*** – existující kontejnery mohou být využité jako šablony.
- **Sdílení kontejnerů** – na základě registru kontejnerů.

Dalším efektem je zvýšení ***bezpečnosti aplikací***. Napadnutá aplikace v kontejneru nemůže ohrozit další aplikace díky její izolaci. Zároveň ***brání tomu, aby jedna aplikace využívala celý dostupný výkon*** a omezovala ostatní aplikace. (Fritsch, 2016)

5.1 Nástroje

5.1.1 Dockerfile

Kontejner je vytvořený pomocí Dockerfile, což je soubor, který obsahuje instrukce pro Docker Engine jak postupovat při vytváření kontejneru pro danou aplikaci.

5.1.2 Docker image

Docker image (obraz) ***obsahuje kód aplikace a nástroje, knihovny a náležitosti*** aplikace. Spuštěním image se ***vytvoří jedna instance kontejneru***. Image je možné vytvořit pomocí Dockerfile nebo ho stáhnout z veřejných repozitářů, např. Docker Hub. Image jsou ***tvořeny z vrstev***, každá z nich koresponduje s verzí obrazu. Při vytvoření kontejneru z image dochází k vytvoření další vrstvy, tzv. kontejnerové. Tato vrstva slouží pro změny vykonávané v době aktivity kontejneru. Změny existují po dobu existence existuje kontejner. Může tak být ***několik současně vytvořených kontejnerů se společnou image***. (Docker, 2020).

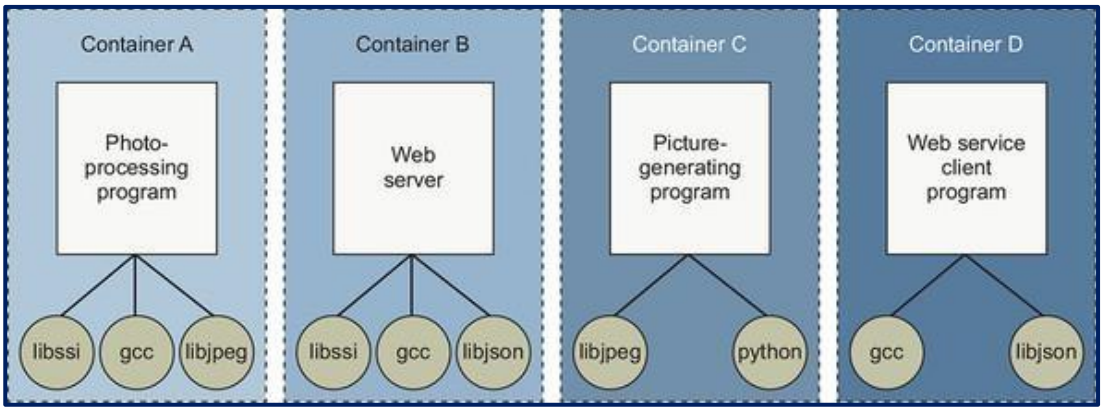
5.1.3 Docker Hub

Docker Hub je ***veřejný repozitář Docker image*** (Docker, 2020). Obsahuje přes ***3 milióny*** kontejnerových image. Všichni uživatelé mohou sdílet svoje obrazy pro kontejnery.

5.2 Přínosy

5.2.1 Organizace

Aplikace jsou postaveny na mnoha různých vazbách a vytvářejí složitou síť interakcí přes celý operační systém. Docker ***udržuje organizaci díky izolaci mezi jednotlivými kontejnery a obrazy*** a proto, že každý kontejner obsahuje všechno, co aplikace potřebuje (Obrázek 5-1). (Nickoloff, 2019).



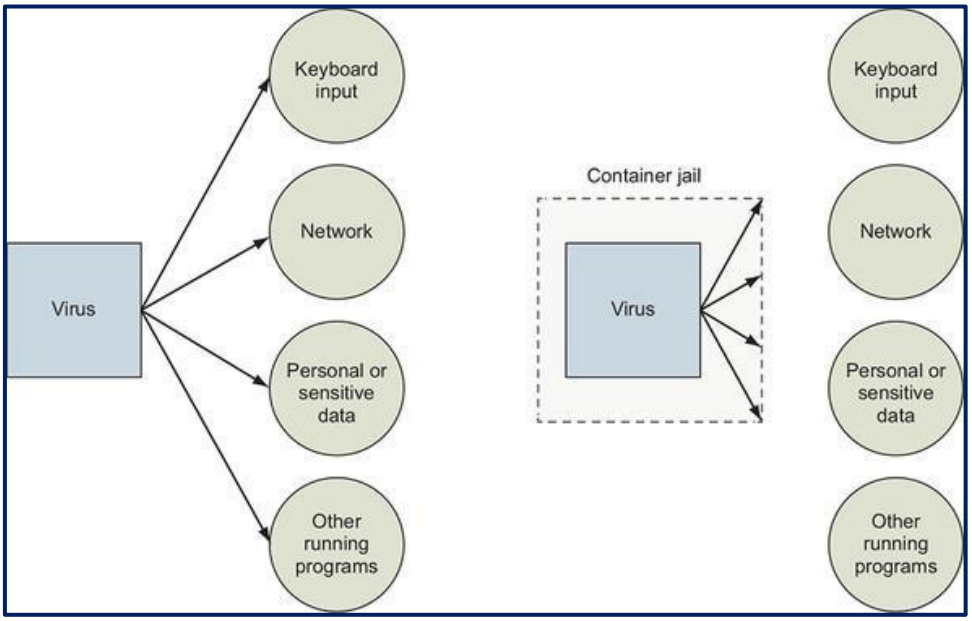
Obrázek 5-1: Organizace procesů v kontajnerech (Nickoloff, 2019)

5.2.2 Přenositelnost

Problém je i v závislosti aplikace na určitém operačním systému a knihovnách. Docker běží jako nativní aplikace v Linuxu a pro macOS a Windows využívá virtuální stroj, v kterém spouští kontejnery. To pomáhá ***provozovat identický software v každém systému***. Docker poskytuje ***konzistentní prostředí pro provoz*** software. Nemusí se zabývat podporou pro různé operační systémy. (Nickoloff, 2019)

5.2.3 Bezpečnost

Kontejnery napomáhají i se zabezpečením aplikací tím, že lze přistupovat pouze k tomu, co se v nich nachází (Nickoloff, 2019). Kontejner zamezí rozšíření viru mimo hranice kontejneru Virus nemá dosah na síť, osobní data nebo k ostatním aplikacím.



Obrázek 5-2: Dopad víru spuštěného přímo v OS a v kontejneru (Nickoloff, 2019)

6. Kubernetes

Podle (Aschmann, 2020, upraveno)

Kubernetes (zkratkou K8s) je open-source platforma pro orchestraci kontejnerů pomocí automatizace nasazování, škálování a správy služeb přes více strojů. Táto technologie byla vyvinutá Googlem (Kubernetes, 2015).

Obrázek 6-1: Logo Kubernetes (Kubernetes, 2020)

Kubernetes se stal jedním z nejpopulárnějších open-source projektů na světě a je považovaný za ***standard vytváření nativních cloudových aplikací***. Je to ***infrastruktura pro distribuované systémy***. (Burns, 2019). Kubernetes naplňuje potřeby provozování kontejnerů ***pomocí těchto praktik***: (Saito, 2019)

- Nasazování kontejnerů
- Trvalé úložiště
- Monitorování stavu kontejnerů
- Správa výpočetních zdrojů
- Automatizované škálování
- Vysoká dostupnost

Využívaním Kubernetes je ***správa kontejnerů zjednodušená***. Úlohy jako nasazení nové verze kontejneru nebo návrat ke staré verzi se řeší jedním příkazem. Kontejner může být automatizovaně ***zastavený nebo odstraněný*** při problémech, např. když přestane odpovídat. V takových situacích se Kubernetes automaticky postará, aby ***aplikace zůstala dostupná*** podle předem stanovených parametrů. Na základě metrik situaci vyhodnotí, vytvoří nový kontejner a nahradí jím nefunkční.

6.1 Komponenty Kubernetes

6.1.1 Pod

Pod je nejmenší a nejjednodušší ***základní jednotka Kubernetes*** modelu, kterou je možné vytvořit nebo nasadit a ***reprezentuje provozovaný proces*** v clusteru. Pod je prostředí pro instanci aplikace v Kubernetes, která se může ***skládat z jednoho nebo více kontejnerů***, které spolupracují a mezi sebou sdílejí zdroje. Kontejnery, které běží na stejném podu, jsou vždy umístěné na stejném uzlu. Pro řízení stavu podů Kubernetes využívá správce. (Saito, 2019).

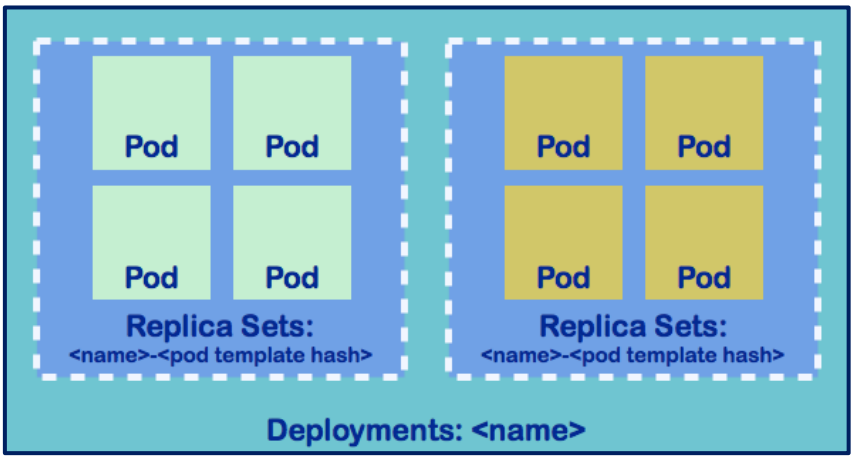
Pokud aplikace potřebuje více procesů, ***každý proces je izolovaný do vlastního kontejneru*** a všechny jsou spouštěné v rámci stejného podu a tak se všem procesům poskytne téměř stejné prostředí, jako kdyby byly v jednom kontejneru. (Burns, 2019; Lukša, 2018)

6.1.2 ReplicaSet

Někdy je třeba provozovat více instancí stejných podů. Spouštění více podů najednou. ***ReplicaSet se používá namísto manuálního vytváření a správy více podů***. V případě selhání podu se ReplicaSet také stará o nápravu. (Burns, 2019; Lukša, 2018; Saito, 2019).

6.1.3 Deployment

Deployment umožňuje ***nasazení nových podů, postupné aktualizace a návraty*** k k jejich původním verzím. Nasazení nové verze aplikace ***se zjednodušuje*** díky postupnému procesu nasazování podů podle definovaných parametrů. Pokud by se vyskytly chyby, proces aktualizace se zastaví. S postupným procesem je tak možné ***přecházet na nové verze aplikací bez jejich odstávky***. Proces je spravovaný ovladačem nasazování, který je součástí Kubernetes clusteru. (Burns, 2019; Saito, 2019). Obrázek ukazuje vztah mezi pody, ReplicaSets a jednotlivými nasazeními. Není tak potřeba manuálně zajišťovat pody nebo ReplicaSets.



Obrázek 6-2: Vztah mezi Pody, ReplicaSets a Deployments (Saito, 2019)

6.1.4 Service

Service je vrstva Kubernetes pro ***zasílání komunikace k podům na základě vazby mezi nasazením a službou.*** Vytváří jednotný a vstupní bod pro pody, které patří ke stejné službě. Uživatelé vytvářejí spojení k dané službě, která směřuje komunikaci na backend k podům. Služby umožňují jejich separaci a zároveň jim poskytují komunikaci na bázi protokolů TCP, UDP a SCTP. (Kubernetes, 2020).

6.1.5 Namespace

Namespace v Kubernetes ***definuje izolované virtuální prostory*** v rámci clusteru. Každý objekt může být součástí pouze jednoho namespace. Namespaces vytvářejí více virtuálních clusterů. (Kubernetes, 2020).

6.2 Cluster

Kubernetes cluster ***obsahuje dva typy strojů*** – ***master***, hlavní uzel, který kontroluje a plánuje všechny aktivity, a ***pracovní uzly***, které je vykonávají.

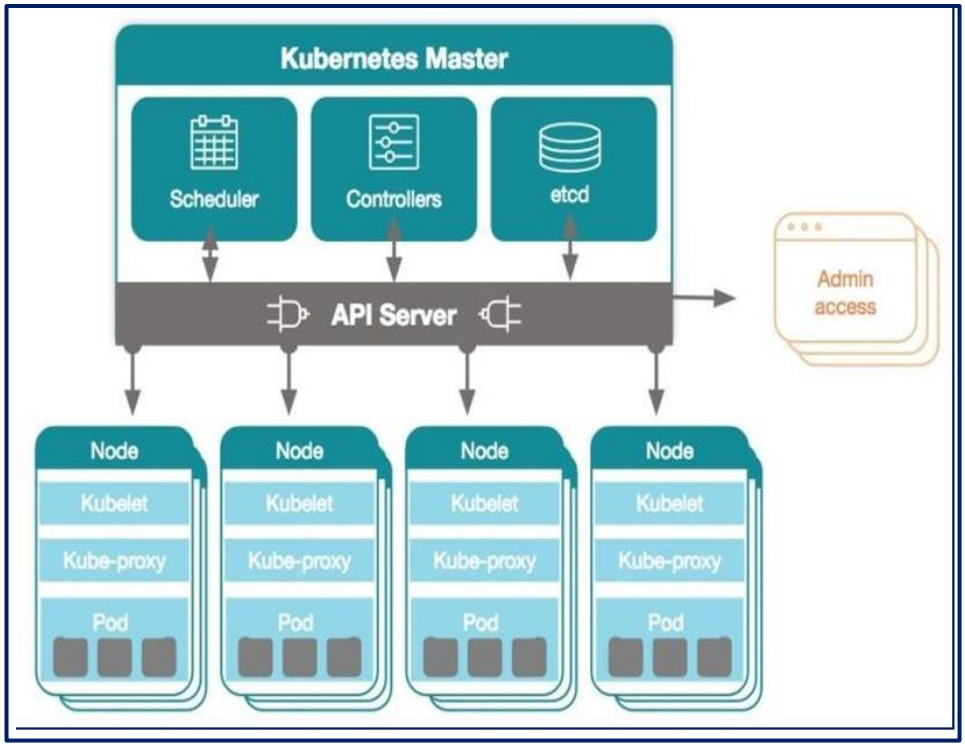
6.2.1 Master

Master se stará o ***kontrolu a řízení uzlů*** v celém clusteru. Pro správu pracovních uzlů využívá tyto komponenty (Mohamed, 2019):

- ***API Server*** – slouží jako front-end pro celý cluster. Externí komunikace s clusterem prochází přes API Server.
- ***Controller Manager*** – provozuje ovladače pro provozovaný cluster a spravuje jeho stav.
- ***Scheduler*** – plánuje aktivity pro pracovní uzly na základě událostí uložených v etcd a rovněž obsahuje rozdělení zdrojů uzlů.
- ***Etcd*** – databáze stavu clusteru.

6.2.2 Uzel

Uzel je pracovní stroj v Kubernetes, kterým ***může být virtuální nebo fyzický stroj***. Každý uzel je spravovaný master uzlem a obsahuje komponenty, které jsou potřebné pro funkce podů. Mezi komponenty pracovního uzlu patří ***prostředí pro spouštění kontejnerů, kubelet a kube-proxy.*** (Kubernetes, 2020).



Obrázek 6-3: Architektura Kubernetes clusteru (Gerrard, 2019)

6.3 Efekty Kubernetes

K ***efektům*** Kubernetes paří např.:

- ***Rychlost –*** vyhodnocuje se podle počtu funkcionalit, které byly doručené s udržením vysoké dostupnosti služby. (Burns, 2019).
- ***Škálování*** – využívaní kontejnerů znamená, že každou změnou dochází k vytvoření nového obrazu, který nahradí starý obraz. Díky tomu se předchází malým, inkrementálním změnám v rámci celé infrastruktury a vede ke zlepšení řízení konfigurací.
- ***Deklarativní konfigurace*** – je alternativou k imperativní konfiguraci, kde se požadovaný stav definuje sérií instrukcí. Imperativní konfigurace stanovuje aktivity a deklarativní konfigurace definuje stav. (Burns, 2019).

7. Zdroje

ALEXANDER, Christopher, Sara ISHIKAWA a Murray SILVERSTEIN, 1977. *A pattern language: towns, buildings, construction*. New York: Oxford University Press. ISBN 978- 0-19-501919-3.

ALTEXSOFT, 2018. DevOps: Principles, Practices, and DevOps Engineer Role. AltexSoft [online]. Dostupné z: https://www.altexsoft.com/blog/engineering/devops-principles- practices-and-devops-engineer-role/

AMAZON, 2020. Cluster Autoscaler - Amazon EKS [online]. Dostupné z: https://docs.aws.amazon.com/eks/latest/userguide/cluster-autoscaler.html

ATLASSIAN, nedatované. Continuous integration vs. continuous delivery vs. continuous deployment. Atlassian [online]. Dostupné z: https://www.atlassian.com/continuous- delivery/principles/continuous-integration-vs-delivery-vs-deployment

BESTPRACTICE, nedatované. Vlastnosti prostredia DevOps [online]. Dostupné z: https://www.bestpractice.sk/sk/Best-practice/DevOps/Vlastnosti-prostredia- DevOps.alej

BMC, nedatované. The Role of Automation in DevOps [online]. Dostupné z: https://www.bmc.com/blogs/automation-in-devops/

BURNS, Brendan, Kelsey HIGHTOWER a Joe BEDA, 2019. Kubernetes: up and running: dive into the future of infrastructure. ISBN 978-1-4920-4653-0.

CALCOTE, Lee, 2018. The Enterprise Path to Service Mesh Architectures. O'Reilly Media, Inc. ISBN 978-1-4920-4178-8.

CLARK, Kim, 2016. Microservices, SOA, and APIs: Friends or enemies? IBM Developer [online]. Dostupné z: https://developer.ibm.com/depmodels/cloud/tutorials/1601_clark- trs/

COSTELLO, Katie, 2019. The Secret to DevOps Success. Gartner [online]. Dostupné z: https://www.gartner.com/smarterwithgartner/the-secret-to-devops-success/

DANIELS, Jennifer Davis. Katherine, 2016. Effective DevOps. O'Reilly Media, Inc. ISBN 978-1-4919-2629-1.

DOCKER, 2020. Docker Trusted Registry overview. Docker Documentation [online]. Dostupné z: https://docs.docker.com/ee/dtr/

DOCKER, 2020. Universal Control Plane overview. Docker Documentation [online]. Dostupné z: https://docs.docker.com/ee/ucp/

FOOTE, Brian a Joseph YODER, 1999. Big Ball of Mud [online]. Dostupné z: http://www.laputan.org/mud/

FOWLER, Martin, 2006. Continuous Integration. martinfowler.com [online]. Dostupné z: https://martinfowler.com/articles/continuousIntegration.html

FRITSCH, Joerg, 2016. Can you operationalize Docker containers? Gartner Blog Network [online]. Dostupné z: https://blogs.gartner.com/joerg-fritsch/can-you-operationalize- docker-containers/

FULTON, Scott, 2015. What Led Amazon to its Own Microservices Architecture. The New Stack [online]. Dostupné z: https://thenewstack.io/led-amazon-microservices- architecture/

GARTNER, 2019. Devops. Gartner [online]. Dostupné z: https://www.gartner.com/en/information-technology/glossary/devops

GERRARD, Ali, 2019. What Is Kubernetes? An Introduction to the Container Orchestration Platform. New Relic Blog [online]. Dostupné z: https://blog.newrelic.com/engineering/what-is-kubernetes/

GOOGLE, 2020. Cluster autoscaler. Google Cloud [online]. Dostupné z: https://cloud.google.com/kubernetes-engine/docs/concepts/cluster-autoscaler?hl=cs

HÁMORI, Ferenc, 2016. How Enterprises Build Microservices Architectures. RisingStack Engineering - Node.js Tutorials & Resources [online]. Dostupné z: https://blog.risingstack.com/how-enterprises-benefit-from-microservices- architectures/

IBM CLOUD EDUCATION, 2019. Microservices [online]. Dostupné z: https://www.ibm.com/cloud/learn/microservices

IBM CLOUD EDUCATION, 2019. Service-oriented Architecture [online]. Dostupné z: https://www.ibm.com/cloud/learn/soa

IBM CLOUD EDUCATION, 2020. What is Docker? [online]. Dostupné z: https://www.ibm.com/cloud/learn/docker

ISTIO, 2020. Visualizing Metrics with Grafana. Istio [online]. Dostupné z: https://istio.io/docs/tasks/observability/metrics/using-istio-dashboard/

JAEGER, nedatované. Jaeger: open source, end-to-end distributed tracing [online]. Dostupné z: https://www.jaegertracing.io/

KRIEF, Mikael, 2019. Learning DevOps [online]. Packt Publishing. ISBN 978-1-83864- 853-4.

KUBERNETES, 2015. Borg: The Predecessor to Kubernetes [online]. Dostupné z: https://kubernetes.io/blog/2015/04/borg-predecessor-to-kubernetes/

KUBERNETES, 2020. Concepts [online]. Dostupné z: https://kubernetes.io/docs/concepts/

KUBERNETES, 2020. Production-Grade Container Orchestration [online]. Dostupné z: https://kubernetes.io/

KWIECIEN, 2019. 10 companies that implemented the microservice architecture and paved the way for others. Divante.com Blog [online]. Dostupné z: https://divante.com/blog/10- companies-that-implemented-the-microservice-architecture-and-paved-the-way-for- others/

LUKŠA, Marko, 2018. Kubernetes in action. Shelter Island, NY: Manning Publications Co. ISBN 978-1-61729-372-6.

MAUERSBERGER, Laura, 2017. Why Netflix, Amazon, and Apple Care About Microservices [online]. Dostupné z: https://www.leanix.net/en/blog/why-netflix-amazon- and-apple-care-about-microservices

MBI, 2016. Terminologický slovník. Management Byznys Informatiky [online]. Dostupné z: https://mbi.vse.cz/mbi/files/termdictionary.docx

MCLUCKIE, Craig, 2016. From Google to the world: the Kubernetes origin story. Google Cloud Blog [online]. Dostupné z: https://cloud.google.com/blog/products/gcp/from- google-to-the-world-the-kubernetes-origin-story/

MICROSOFT, 2019. Use the cluster autoscaler in Azure Kubernetes Service (AKS) [online]. Dostupné z: https://docs.microsoft.com/en-us/azure/aks/cluster-autoscaler

MIRANDA, George, 2018. The Service Mesh. O'Reilly Media, Inc. ISBN 978-1-4920-3131- 4.

MOHAMED, Rania, 2019. Kubernetes Cluster vs Master Node. SUSE Communities [online]. Dostupné z: https://www.suse.com/c/kubernetes-cluster-vs-master-node/

MOY, Edward, 2019. Devops Tools of the Trade. OsoLabs.com [online]. Dostupné z: https://www.osolabs.com/blog/devops-tools-of-the-trade/

NEMER, Joe, 2019. Advantages and Disadvantages of Microservices Architecture [online]. Dostupné z: https://cloudacademy.com/blog/microservices-architecture-challenge- advantage-drawback/

NEWMAN, Sam, 2015. *Building microservices: designing fine-grained systems*. First Edition. Beijing Sebastopol, CA: O'Reilly Media. ISBN 978-1-4919-5035-7.

NICKOLOFF, Jeff, Stephen KUENZLI a Bret FISHER, 2019. Docker in action. ISBN 978-1- 61729-476-1.

RED HAT, 2018. What's a service mesh? [online]. Dostupné z: https://www.redhat.com/en/topics/microservices/what-is-a-service-mesh

RICHARDSON, Chris, 2019. Microservices Pattern: Sagas. microservices.io [online]. Dostupné z: http://microservices.io/patterns/data/saga.html

RICHARDSON, Chris, 2019. *Microservices patterns: with examples in Java*. Shelter Island, New York: Manning Publications. ISBN 978-1-61729-454-9.

SAITO, Hideto, Hui-Chuan LEE, Cheng-Yang WU a an O'Reilly Media Company SAFARI, 2019. DevOps with Kubernetes - Second Edition. ISBN 978-1-78953-399-6.

SHANTALA, 2018. Microservices – the Evolution. One curious mind [online]. Dostupné z: http://www.shantala.io/microservices-the-evolution/

SMARTBEAR, 2015. Why You Can't Talk About Microservices Without Mentioning Netflix. Smart-Bear.com [online]. Dostupné z: https://smartbear.com/blog/develop/why-you-cant- talk-about-micro-services-without-ment/

SMITH, Floyd a Owen GARRETT, 2018. What Is a Service Mesh? NGINX [online]. Dostupné z: https://www.nginx.com/blog/what-is-a-service-mesh/

SYSELEVEN, 2019. The history of Kubernetes | Docker, Project Borg & the Future [online]. Dostupné z: https://www.syseleven.de/en/solutions/kubernetes

WATSON, Amy, 2020. Number of Netflix subscribers 2019. Statista [online]. Dostupné z: https://www.statista.com/statistics/250934/quarterly-number-of-netflix-streaming- subscribers-worldwide/